Scalable Access Control in Content-Based Publish-Subscribe Systems

Mudhakar Srivatsa and Ling Liu
College of Computing, Georgia Institute of Technology
{mudhakar, lingliu}@cc.gatech.edu

Abstract. Content-based publish-subscribe (pub-sub) systems are an emerging paradigm for building a large number of distributed systems. Access control in a pub-sub system refers to secure distribution of events to clients subscribing to those events without revealing its secret attributes to the unauthorized subscribers. To provide confidentiality guarantees the secret attributes in an event is encrypted so that only authorized subscribers can read them. However, in a content-based pub-sub system, every event can potentially have a different set of authorized subscribers. In the worst case, for NS subscribers, there are 2^{NS} subgroups, and each event can potentially go to a different subgroup. Hence, efficient key management is a big challenge for implementing access control in pub-sub systems. In this paper, we describe efficient and scalable key management algorithms for securely implementing access control rules in pub-sub systems. We ensure that the key management cost is linear in the number of subscriptions and completely independent of the number of subscribers NS. We present a concrete implementation of our proposal on an operational pub-sub system. An experimental evaluation of our prototype shows that our proposal meets the security requirements while maintaining the scalability and performance of the pub-sub system.

Keywords. Publish-Subscribe Systems, Access Control, Key Management, Performance & Scalability

1 Introduction

A growing number of Internet applications require information dissemination across different organizational boundaries, heterogeneous platforms, and a large, dynamic population of publishers and subscribers. A publish-subscribe (pub-sub for short) system enables information dissemination across geographically scattered and potentially unbounded number of publishers and subscribers. In such an environment, publishers publish information in the form of *events* and subscribers have the ability to express their interests in an event or a pattern of events in the form of subscription *filters*.

Access Control. Access control in a pub-sub system mandates that an event is intelligible to a subscriber if and only if the event matches the subscriber's registered interest. To provide confidentiality guarantees the secret attributes in an event is encrypted so that only authorized subscribers can read them. However, access control in pub-sub systems differ from access control in a traditional group key management protocols in several ways. A pub-sub system defines access control on publications using complex subscriptions. For example, publications on confidential medical records about a patient whose age is 25 should only be intelligible to a subscriber with the following subscription: 'notify me updates on medical records about all patients whose age is greater than 20'; but not by a subscriber with the following subscription: 'notify me updates on medical records about all patients whose age is greater than 30'.

Given a flexible subscription model, every event can potentially go to a different subset of subscribers thereby making efficient implementation of confidentiality very difficult. In the worst case, for NS subscribers, there are 2^{NS} subgroups, and each event can potentially go to a different subgroup. With thousands of subscribers it is infeasible to setup static security groups for every possible subgroup. Opyrchal and Prakash [18] provides a more comprehensive discussion on the infeasibility of using group key management protocols like broadcast encryption [9], secure multicast GKMP [12, 11] and logical key hierarchies [21] in the context of content-based pub-sub networks. Opyrchal and Prakash [18] propose a group key clustering and caching based mechanisms to alleviate the problem of managing possibly exponential number of group keys (2^{NS}).

Approach. Bearing these issues in mind, we propose secure, efficient and scalable key management algorithms for content-based pub-sub networks. Unlike other solutions proposed in the past, our proposal does not associate keys with a subscribers or groups of subscribers, thereby ensuring that the number of keys to be managed is *independent*

of the number of subscribers in the pub-sub system. We associate an authorization key $K(f)$ with a subscription filter f and an encryption key $K(e)$ with an event e. We use the encryption key $K(e)$ to *encrypt* the secret attributes in an event e. A subscriber can *decrypt* these secret attributes using *any* key $K(f)$ if and only if the event e matches the subscription filter f. This is achieved by mapping the authorization key $K(f)$ and the encryption key $K(e)$ to a common key space. We use key derivation algorithms to ensure that $K(e)$ can be efficiently derived from $K(f)$ if the event e matches the filter f; otherwise, guessing $K(e)$ from $K(f)$ is computationally infeasible.

Our approach has two important performance and scalability benefits over the traditional subscriber group [18] based approach. (i) Our approach ensures the number of authorization keys are *independent* of the number of subscribers by using efficient key derivation algorithms. Using our approach the key server maintains only one key and the subscriber maintains a small and constant number of keys per subscription (as against the worst case exponential number of keys using a group key management approach). (ii) Our approach allows the key server to be *stateless* and ensures that the cost of a subscription is a small constant (independent of NS). The subscriber group based approach may require changes to the groups and group keys when a new subscriber joins the system, thereby incurring higher computation, communication and storage costs.

Contributions. In this paper we demonstrate our approach by constructing key spaces for four types of publication-subscription matching: topic or keyword based matching, numeric attribute based matching, category based matching and string based prefix/suffix matching. We formally define these matching semantics in Section 2. We provide secure and efficient key derivation algorithms to guarantee the confidentiality of secret attributes in an event. We enhance the performance of our key derivation algorithms by constructing a semantic key cache. We present a modular and stackable implementation of our proposal on the Siena [6] pub-sub system. Our experiments show that our proposal meets the security goals while maintaining the system's simplicity, scalability and performance metrics.

Paper Outline. The rest of this paper is organized as follows. Section 2 presents a reference pub-sub model and formally defines various content-based matching semantics. Section 3 presents the concrete algorithms for implementing access control followed by a semantic key caching technique to enhance the performance of our algorithms. Section 4 presents a sketch of our implementation on the Siena pub-sub system followed by a detailed performance evaluation. We discuss related work in Section 5 and finally conclude in Section 6.

2 Preliminaries

2.1 Reference Pub-Sub Model

We use a reference pub-sub model that is very similar to a content-based pub-sub system like Siena [6]. Publications are specified in terms of events and subscriptions are expressed in terms of filters. Formally, an event e is denoted as $e = \langle name, value \rangle^*$, where $name$ refers to some attribute name, $value$ corresponds to its published value, and the notation * indicates that an event may comprise of one or more name and value tuples. A filter f is denoted as $f = \langle name, operator, value \rangle^*$, where $name$ refers to some attribute name, $value$ specifies an attribute value, $operator$ refers to a binary operator, and the notation * indicates that a filter may comprise of one or more constraints in a conjunctive form. Operators typically include common equality and ordering relations ($=$, $<$, $>$, etc) for numeric attributes; and substring, prefix, suffix operators for strings. An event e is said to *match* a filter f if e satisfies all the constraints in the filter f. In a pub-sub system disseminating confidential medical records, an example event could be: $e = \langle\langle \texttt{topic}, \texttt{cancerTrail} \rangle, \langle \texttt{age}, 25 \rangle, \langle \texttt{patientRecord}, record \rangle\rangle$. An example subscription could consist of the following constraints: $f = \langle\langle \texttt{topic}, EQ, \texttt{cancerTrail} \rangle, \langle \texttt{age}, >, 20 \rangle\rangle$, where EQ denotes the keyword based matching operator and $>$ denotes the greater-than numeric operator.

Pub-Sub system uses a third party authorization service to restrict the set of events that can be read by a subscriber. Access control in a pub-sub system is specified at the granularity of a subscription filter. A subscriber receives one authorization permit for every subscription that it is allowed to read. An authorization for a subscription filter is associated with a lifetime. We use an epoch based approach wherein all authorization permits are valid for one time epoch. At the end of an epoch, the subscriber will have to obtain a new authorization permit to read events that match the subscription filter in the next epoch. Using an epoch based subscription model allows the publishers to charge its subscribers on a periodic basis.

Pub-Sub systems may use a wide-variety of communication infrastructures to disseminate events from a group of publishers to a group of subscribers ranging from: direct one-to-one channels, multicast networks and overlay networks. Our proposal is independent of the communication infrastructure. However, our implementation is based on Siena, which uses an overlay network based communication infrastructure for scalable event dissemination.

2.2 Threat Model

We assume an honest-but curious model for both the publishers and the subscribers (discretionary access control model). A curious publisher may be interested in reading the events published by other publishers. For subscribers, we define authorization on a per subscription basis and a subscription epoch. A subscriber S is authorized to read an event e if the event e matches one of its active subscriptions. We assume that a subscriber S who is authorized to read an event e does not reveal its contents to other unauthorized subscribers (otherwise, this would be equivalent to solving the digital copyrights problem). However, unauthorized subscribers may be curious to read those events that do not match their subscriptions. We assume that the communication infrastructure used for event dissemination may not be secure. Hence, malicious subscribers may *eavesdrop* on the pub-sub messages sent through the communication infrastructure.

3 Key Management for Publish-Subscribe Systems

3.1 Overview

We use authorization keys and encryption keys to support access control on pub-sub systems. These keys serve complementary purposes. An encryption key is used to maintain the confidentiality of an event from subscribers who have not subscribed to that event. An authorization key is designed to encode content-based matching semantics into a key derivation algorithm such that an authorized subscriber can efficiently derive the encryption keys for those events that match their subscriptions. In this paper, we demonstrate our approach using four different types of publication-subscription matching: topic or keyword based matching, numeric attribute based matching, category based matching, and string based suffix/prefix matching.

For topic or keyword based matching, an authorization key $K(f)$ associated with a filter f = $\langle$`topic`, EQ, `cancerTrail`$\rangle$ must be capable of decrypting the message msg in event e = $\langle\langle$`topic`, `cancerTrail`$\rangle$, $\langle$`message`, $msg\rangle\rangle$. On the other hand, a key $K(f')$ associated with filter f' = $\langle$`topic`, EQ, `humanGenome`$\rangle$ should not be able to decrypt msg in event e. For numeric attribute based matching, a key $K(f_1)$ used for the filter f_1 = $\langle\langle$`topic`, EQ, `cancerTrail`$\rangle$, $\langle$`age`, $>$, 20$\rangle\rangle$ and a key $K(f_1')$ used for the filter f_1' = $\langle\langle$`topic`, EQ, `cancerTrail`$\rangle$, $\langle$`age`, $>$, 30$\rangle\rangle$ must be capable of decrypting the message msg in event e_1 = $\langle\langle$`topic`, `cancerTrail`$\rangle$, $\langle$`age`, 35$\rangle$, $\langle$`message`, $msg\rangle\rangle$. On the other hand, key $K(f_1)$ should be capable of decrypting the message msg in event e_1' = $\langle\langle$`topic`, `cancerTrail`$\rangle$, $\langle$`age`, 25$\rangle$, $\langle$`message`, $msg\rangle\rangle$, but not the key $K(f_1')$. For category based matching, a key $K(f_2)$ used for filter f_2 = $\langle\langle$`topic`, EQ, `cancerTrail`$\rangle$, $\langle$`news`, $\ni$, `unclassifiedNews`$\rangle\rangle$, a key $K(f_2')$ used for f_2' = $\langle\langle$`topic`, EQ, `cancerTrail`$\rangle$, $\langle$`news`, $\ni$, `classifiedNews`$\rangle\rangle$, and a key $K(f_2'')$ used for f_2'' = $\langle\langle$`topic`, EQ, `cancerTrail`$\rangle$, $\langle$`news`, $\ni$, `secretNews`$\rangle\rangle$ must be capable of decrypting the event e_2 = $\langle\langle$`topic`, `cancerTrail`$\rangle$, $\langle$`news`, `unclassifiedNews`$\rangle$, $\langle$`message`, $msg\rangle\rangle$. On the other hand, only $K(f_2'')$ should be capable of decrypting the message msg in e_2' = $\langle\langle$`topic`, `cancerTrail`$\rangle$, $\langle$`news`, `secretNews`$\rangle$, $\langle$`message`, $msg\rangle\rangle$, but not the keys $K(f_2)$ and $K(f_2')$. For string based prefix/suffix matching, a key $K(f_3)$ used for filter f_3 = $\langle\langle$`topic`, EQ, `cancerTrial`$\rangle$, $\langle$`name`, PF, `a`$\rangle\rangle$ and a key $K(f_3')$ used for the filter f_3' = $\langle\langle$`topic`, EQ, `cancerTrial`$\rangle$, $\langle$`name`, PF, `an`$\rangle\rangle$ should be capable of decrypting the message msg in event e_3 = $\langle\langle$`topic`, `cancerTrial`$\rangle$, $\langle$`name`, `andy`$\rangle$, $\langle$`message`, $msg\rangle\rangle$. On the other hand, only $K(f_3)$ must be capable of decrypting the message msg in e_3' = $\langle\langle$`topic`, `cancerTrial`$\rangle$, $\langle$`name`, `alex`$\rangle$, $\langle$`message`, $msg\rangle\rangle$, but not the key $K(f_3')$.

In the following sections, we demonstrate our approach using these four popular types of publication-subscription matching: topic or keyword based matching, numeric attribute based matching, category based matching, and string based prefix/suffix matching. In particular we describe algorithms to derive $K(f)$ for a subscription filter f and $K(e)$ for encrypting an event e such that the derivation satisfies the matching semantics. We first describe our techniques to handle simple subscriptions that consists of a topic and at most one constraint, say, f = $\langle\langle$`topic`, EQ, `cancerTrail`$\rangle$, $\langle$`age`, $>$, 15$\rangle\rangle$. A complex subscription could consist of constraints combined using the $\wedge$ and $\vee$

Boolean operators. We present algorithms to handle simple subscriptions in Sections 3.2, 3.3, 3.4 and 3.5 followed by techniques to handle complex subscriptions in Section 3.6.

3.2 Topic or Keyword Based Matching

We first describe how we implement access control on topic/keyword based matching. Our techniques for topic or keyword based matching are very simplistic. However, numeric attribute based matching, category based matching and string based matching significantly differ in the techniques employed to generate the authorization keys and the encryption keys.

Subscription. Given a simple topic based subscription f the subscription key is generated by an authorization service *MS* as:

$$\begin{aligned} f &= \langle \texttt{topic}, EQ, w \rangle \\ K(f) &= K(w) = KH_{rk(MS)}(w), KS(w) = KH^2_{rk(MS)}(w) \end{aligned}$$

Note that $rk(MS)$ denotes the 128-bit *MS*'s secret key and $KH_{SK}(w)$ denotes a keyed hash of string w using a keyed-pseudo random function (PRF) KH (approximated by HMAC-MD5 or HMAC-SHA1 [13]) and a secret key SK. Also, KH^j denotes j successive applications of the PRF KH.

A subscription key for a topic w with one keyword based matching constraint is constructed as follows. Subscriptions on a topic w with a numeric attribute based constraint, category and string prefix/suffix based constraint are discussed in Section 3.3, 3.4, and 3.5.

$$\begin{aligned} f &= \langle \langle \texttt{topic}, EQ, w \rangle, \langle name, EQ, value \rangle \rangle \\ K(f) &= K^{name}_{value} = KH_{K(w)}(name \parallel value) \rangle, KS(w) = KH_{rk(MS)}(w) \end{aligned}$$

Note that $\parallel$ denotes string concatenation. For example, given a subscription filter *f* = $\langle \langle$ `topic`, EQ, `cancerTrail` $\rangle$, $\langle$ `gender`, EQ, `F` $\rangle \rangle$ the authorization key is computed as $K^{\texttt{gender}}_{\texttt{F}} = KH_{K(\texttt{cancerTrail})}(\texttt{gender} \parallel \texttt{F})$, where $K(\texttt{cancerTrail}) = KH_{rk(MS)}(\texttt{cancerTrail})$.

Advertisement. Similar to the subscription process, a publisher P who wishes to publish events under a topic w obtains an authorization permit from the authorization service *MS*. Given a simple topic based subscription f the subscription key is generated by an authorization service *MS* as:

$$\begin{aligned} f &= \langle \texttt{topic}, EQ, w \rangle \\ K(f) &= \langle K(w) = KH_{rk(MS)}(w), KP(P, w) = KH_{KS(w)}(P) \rangle \end{aligned}$$

The authorization key $K(w)$ is constructed by the *MS* using the same technique as that of subscription. The key $KP(P, w)$ is derived as $KP(P, w) = KH_{KS(w)}(P)$, where $KS(w) = KH^2_{rk(MS)}(w)$. The key $KP(P, w)$ is used to disallow one publisher from being able to read events published by other publishers on the same topic. Note that using $KP(P, w)$ a publisher P cannot guess $KP(P', w)$ corresponding to another publisher P'. However, the key $KS(w)$ allows a subscriber to derive $KP(P, w)$ for all publishers and thus read all events published under topic w irrespective of its publisher. Note using the subscriber group approach achieving the same goal would have increased the number of keys per subscriber and the load on key server by a factor $|P|$, where $|P|$ denotes the number of publishers. Our approach reduces the key management overhead and pays a very small cost for deriving $KP(P, w)$ from $KS(w)$. Constructing an authorization permit for a publisher to publish events on a topic with one keyword based matching constraint is very similar to that used for constructing subscriptions (discussed above).

Publication. A publisher P constructs the encryption keys for events as follows:

$$\begin{aligned} e &= \langle \langle \texttt{publisher}, P \rangle, \langle \texttt{topic}, w \rangle, \langle \texttt{message}, msg \rangle \rangle \\ K(e) &= K(w) \oplus KP(P, w) \end{aligned}$$

One can use any standard symmetric key encryption algorithm (say, DES [10] or AES [17]) for E. Observe that any subscriber who has subscribed for topic w possesses the authorization keys $K(w)$ and $KS(w)$. The authorized

subscriber first derives $KP(P, w)$ from $KS(w)$ and P. Then, it uses the authorization key $K(w)$ and the derived key $KP(P, w)$ to derive the encryption key $K(e)$.

A publication on a topic w with one keyword based matching constraint is constructed as follows.

$$\begin{aligned} e &= \langle\langle \texttt{publisher}, P\rangle, \langle \texttt{topic}, w\rangle, \langle name, value\rangle, \langle \texttt{message}, msg\rangle\rangle \\ K(e) &= K_{value}^{name} \oplus KP(P, w) \end{aligned}$$

For example, given an event e = $\langle\langle$publisher, $P\rangle$, $\langle$topic, cancerTrail$\rangle$, $\langle$gender, F$\rangle$, $\langle$message, $msg\rangle\rangle$ its encryption key $K(e) = K_{\texttt{F}}^{\texttt{gender}} \oplus KP(P, w)$, where $K_{\texttt{F}}^{\texttt{gender}} = KH_{K(\texttt{cancerTrail})}(\texttt{gender} \parallel \texttt{F})$ and $K(\texttt{cancerTrail}) = KH_{rk(MS)}(\texttt{cancerTrail}$. Observe that a subscriber who has subscribed for a filter f = $\langle\langle$topic, EQ, cancerTrail$\rangle$, $\langle$gender, EQ, F$\rangle\rangle$ has the authorization key $KH_{K(\texttt{cancerTrail})}(\texttt{gender} \parallel \texttt{F})$. The subscriber can use this authorization key to derive the encryption key for the event e.

Unsubscription and Unadvertisement by Rekeying. Using an epoch based subscription model permits us to revoke subscriptions using periodic rekeying. We periodically change all authorization keys by changing *MS*'s secret key $rk(MS)$. Observe that since all the authorization keys are derived from the *MS*'s secret key, changing the key $rk(MS)$ changes the values for all the authorization keys. Concretely, we perform a periodic rekeying operation as follows. We divide time into epochs of $epoch$ time units (say, one month). All subscriptions and advertisements need to be renewed at the beginning of every time epoch. We number epochs with consecutive integers starting from epoch number 0. The secret key used by the *MS* in the T^{th} epoch is derived from the secret key $rk(MS)$ as $rk(MS, T) = KH_{rk(MS)}(T)$. The *MS* uses $rk(MS, T)$ to replace $rk(MS)$ during the T^{th} epoch for generating the authorization keys. Hence, if a subscriber S unsubscribes from topic w in epoch T, it would still be able to read the contents of publications under topic w till the end of epoch T (but not after epoch T) by eavesdropping on the pub-sub communication infrastructure. Unsubscription and unadvertisements by rekeying additionally facilitates the authorization service *MS* to collect periodic subscription and advertisement fee.

3.3 Numeric Attribute Based Matching

In this section, we present our key derivation algorithm for numeric attribute based constraints supports range queries on numeric attributes. Given a numeric attribute, say age, we can construct a subscription filter f = $\langle$age, $\in$, $(l, u)\rangle$. The filter f matches any event e = $\langle$age, $v\rangle$ if and only if $l \leq v \leq u$, that is, the value v belongs to the range (l, u). We associate an authorization key $K(f)$ with every subscription filter f and an encryption key $K(e)$ with every event e. The authorization keys and the encryption keys satisfy the following properties:

- Given $K(f)$ it should be computationally easy to derive a key $K(e)$, if $v \in (l, u)$, that is, $l \leq v \leq u$.
- Given $K(f)$ it should be computationally hard to derive a key $K(e)$, if $v \notin (l, u)$, that is, $(v < l) \vee (v > u)$.

We construct keys that satisfy the above mentioned properties as follows. We map the authorization keys and encryption keys to the common key space using a numeric attribute key tree (NAKT for short). Given a subscription filter $\langle$age, $\in$, $(l, u)\rangle$, we use the NAKT to derive a small set of authorization keys that corresponds to the range (l, u), denoted by $K_{(l,u)}^{num}$, where num denotes the name of the numeric attribute. The NAKT enables one to easily (computationally) derive a key $K_{(l',u')}^{num}$ from $K_{(l,u)}^{num}$ if and only if $l \leq l' \leq u' \leq u$. For any event e = $\langle$age, $v\rangle$, we encrypt the message with $K(e) = K_{(v,v)}^{num}$. By the construction of the numeric attribute key tree it follows that $K(e)$ is easily derivable from $K(f)$ if and only if $l \leq v \leq u$.

Preliminaries. In the rest of this section, we present our techniques for numeric attribute based matching. We propose a mechanism that is capable of balancing performance and expressiveness of the numeric attribute based matching algorithm. We quantify the trade-off between performance and expressiveness by introducing the concept of a *least count* $lc(num)$ for a numeric attribute num. Let $R(num)$ be the range of a numeric attribute num. Let us suppose that the numeric attribute age takes value from the domain of integers with minimum value zero and maximum value 31. Then we have R(age) = $(0, 31)$. If lc(age) = 4, then only ranges (l, u) where $l \ mod\ 4 \equiv 0$ and $u \ mod\ 4 \equiv 3$ are permitted. Hence, a subscriber may subscribe for the range $(16, 27)$ but not for the range $(17, 29)$. If subscriber

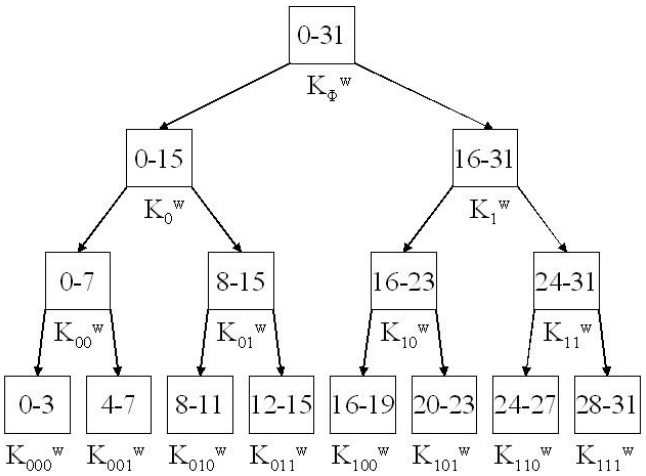

Figure 1: Key Tree: Range Queries on Numeric Attributes

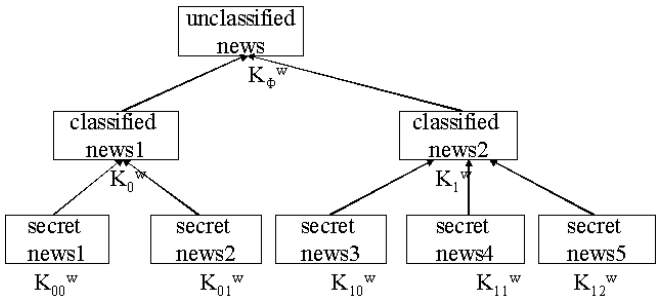

Figure 2: Key Tree: Category Hierarchy

S indeed wants to receive all events covered by the range $(17, 29)$, then S would have to subscribe for a bigger range $(16, 31)$. The higher the value of $lc(num)$, the better is the performance of our key derivation algorithm, but the lower is its expressiveness. Without loss of generality, we assume that the actual range of the numeric attribute num is $(0, |R(num)| - 1)$, where $|R(num)|$ denotes the size of range $R(num)$. Note that if the actual range of the attribute num is $(min(num), max(num))$, then we can always normalize it to $(0, max(num) - min(num))$. Now, we describe a technique to construct a NAKT, and then present the algorithm for constructing subscription and publication using the NAKT. The construction of an advertisement and unsubscription is similar to that of a topic or keyword based matching (section 3.2) and will be omitted here.

Numeric Attribute Key Tree (NAKT). We first present a technique to construct the NAKT. Given a numeric value $v \in (0, |R(num)| - 1)$, we map it to a key tree identifier $ktid$, where $ktid = b_0 b_1 \cdots b_{m-1}$ is the binary representation of the number $\lfloor \frac{v}{lc(num)} \rfloor$ and $m = \log_2(\frac{|R(num)|}{lc(num)})$. The key tree identifiers are arranged in the form of a binary tree with depth m. Figure 1 shows a numeric attribute key tree for $R(num) = (0, 31)$ and $lc(num) = 4$. Each internal element in the tree is assigned a key K_{ktid}^{num} that corresponds to the key tree identifier $ktid$ for the numeric attribute num. The key tree is designed such that given a parent key all its children keys can be easily derived; but the converse is computationally infeasible.

Let the symbol $\emptyset$ (*null*) be used to label the root element of a NAKT. We derive the authorization key for the root element corresponding to the key tree as $K_{\emptyset}^{num} = KH_{K(w)}(num)$, where KH is a keyed pseudo-random function, $K(w) = KH_{rk(MS)}(w)$ is the authorization key for the topic w, and $rk(MS)$ denotes the *MS*'s secret key. An example topic would be w = `cancerTrail` and numeric attribute num = `age`. We derive the key for an internal element with $ktid = \xi \parallel b$ recursively as $K_{\xi \parallel b}^{num} = H(K_{\xi}^{num} \parallel b)$, for some $\xi \in (0+1)^*$, $b = 0$ or 1 and H is a one-way hash function (approximated by MD5 [22] or SHA1 [8]). Note that $\parallel$ denotes string concatenation. For example, K_0^{num} is derived as $K_0^{num} = H(K_{\emptyset}^{num} \parallel 0)$ and K_1^{num} is derived as $K_1^{num} = H(K_{\emptyset}^{num} \parallel 1)$.

Publication. A publisher P constructs the encryption key for a numeric attribute event e as follows:

$$\begin{aligned} e &= \langle\langle \texttt{publisher}, P\rangle, \langle \texttt{topic}, w\rangle, \langle num, v\rangle, \langle \texttt{message}, msg\rangle\rangle \\ K(e) &= K_{ktid(v)}^{num} \oplus KP(P, w) \end{aligned}$$

Note that $ktid(v)$ maps a numeric value v to a key tree identifier as $ktid(v) = int2bin(\lfloor \frac{v}{lc(num)} \rfloor)$. For example, a publication e = $\langle\langle$`publisher`, $P\rangle$, $\langle$`topic`, `cancerTrail`$\rangle$, $\langle$`age`, 22$\rangle$, $\langle$`message`, $msg\rangle\rangle$ is encrypted as follows. P identifies that the leaf node in the NAKT, which contains $v = 22$ has an identifier $ktid(22) = int2bin(\frac{22}{4})$ = $int2bin(5) = 101$ (lc(`age`) = 4, see Figure 1). Hence, P generates the a secure event e constructs the encryption key $K(e) = K_{101}^{\texttt{age}} \oplus KP(P, \texttt{cancerTrail})$.

Subscription. A subscriber can subscribe for any range over the numeric attributes (limited by the least count $lc(num)$). The subscription range may span one or more elements in the NAKT. Given a range (l, u) we identify

the smallest set of elements in the NAKT that spans the range (l, u). For example, the smallest set of elements in the NAKT that spans the range $(8, 19)$ are $(8, 15)$ and $(16, 19)$. In general, given a range (l, u) one can identify the smallest set of elements in the NAKT that spans the range (l, u) by performing a depth first search starting at the root of the NAKT. We first discuss those subscriptions whose subscription range spans exactly one element in the NAKT. We then extend the solution to handle subscriptions whose range spans multiple elements in the NAKT.

Suppose a subscriber S has subscribed for a range (l, u) that maps to exactly one element in the key tree (say $(16, 31)$ in Figure 1). The subscription (f) and its the authorization key $K(f)$ are as shown below. Note that PF denotes the prefix operator such that $PF(value_\alpha, value_\phi)$ is true if and only if $value_\phi$ is a prefix of $value_\alpha$. We construct the modified subscription using an authorization key $K^{num}_{ktid(l,u)}$ as follows.

$$\begin{aligned} f &= \langle\langle \texttt{topic}, EQ, w\rangle, \langle \texttt{num}, \geq, l\rangle, \langle \texttt{num}, \leq, u\rangle\rangle \\ K(f) &= K^{num}_{ktid(l,u)}, KS(w) \end{aligned}$$

Given a publication with key tree identifier equal to $ktid_\alpha$ a subscriber who has subscribed for key tree identifier equal to $ktid_\phi$ does the following. The subscriber checks if $ktid_\phi$ is a prefix of $ktid_\alpha$. If so, the subscriber derives the encryption key $K^{num}_{ktid_\alpha}$ from the authorization key $K^{num}_{ktid_\phi}$. Note that the generation of children keys from its parent's key is computationally efficient because it uses a fast one-way hash function. However, it is computationally infeasible for a subscriber to derive the keys corresponding to its ancestors or its siblings. For example, given a publication with $ktid_\alpha$ = 101, a subscriber who has subscribed for $ktid_\phi$ = 1 decrypts the message msg in a publication as follows. Given $ktid_\alpha$ = 101 and $ktid_\phi$ = 1, the subscriber first extracts the suffix 01. Then, S derives K^{num}_{10} = $H(K^{num}_1 \parallel 0)$ and K^{num}_{101} = $H(K^{num}_{10} \parallel 1)$. Now, S can use K^{num}_{101} to decrypt the message in the publication.

Now suppose that a subscriber S wishes to subscribe for some range that does not correspond to exactly one element in the key tree, say a range $(8, 19)$. We split such a subscription into two subscriptions, one for the range $(8, 15)$ and the other for the range $(16, 19)$. Observe that given a range (l, u) one can identify the smallest set of elements in the NAKT that spans the range (l, u) by performing a depth first search starting at the root of the NAKT. In general, if one uses a a-ary numeric attribute key tree ($a \geq 2$), any range can always be subdivided into no more than $(a - 1)\ \log_a(\frac{|R(num)|}{lc(num)})$ sub-ranges. One can show that this is a monotonically increasing function in a (for $a \geq 2$) and thus has a minimum value when $a = 2$. Hence, a binary key tree is optimal and it requires no more than $\log_2(\frac{|R(num)|}{lc(num)})$ sub-ranges for any given subscription range. This translates to a worst case logarithmic blowup in the size of a subscription. Therefore, a subscriber may have to maintain $O(\log_2(|R(num)|)$ authorization keys for a subscription on a numeric attribute. Note that maintaining one authorization key for every possible subscription on the numeric attribute requires $O(|R(num)|)$ keys. Hence, our key derivation algorithm significantly reduces the number of keys to be managed and yet preserves the confidentiality guarantees provided by the system.

Key Management Cost and Key Derivation Cost Trade-Off. In addition, one can also show that using a a-ary numeric attribute key tree, the average number of ranges into which any randomly chosen range need to be subdivided is $\frac{1}{2}$*$(a - 1)\log_a(\frac{|R(num)|}{lc(num)})$. One can show that this is a monotonically increasing function in a, that is, the number of keys (and thus the cost of key management) increases monotonically with a. On the other hand, as a increases the height of the key tree ($\log_a(\frac{|R(num)|}{lc(num)})$) decreases, that is, the cost of key derivation decreases monotonically with a. Our experiments show that the cost of key derivation is very small. Therefore, we chose to use a binary key tree (a = 2) that minimizes the key management cost.

3.4 Category Based Matching

In this section, we present techniques for access control on named categories that are typically arranged as an ontology tree [27]. In an ontology or a category tree the children of a node represent more detailed information about the same topic than its parent and thus may be considered more confidential. An example category hierarchy that is applicable in a military scenario is shown in Figure 2. A subscriber who subscribes for `secretNews1` is implicitly entitled to receive all publications published under categories `classifiedNews1` and `unclassifiedNews`; but the converse is not true. Additionally, a subscriber who subscribes for `secretNews1` is not permitted to read events categorized under `classifiedNews2`. In general, publications marked `unclassified` contain general (less detailed and less confidential) information than those marked `classified` and `secret`.

Our key derivation algorithm supports category based subscriptions. Given a category, say `news`, we can construct a subscription filter f = $\langle$`news`, $\ni$, $cat\rangle$. The filter f matches any event e = $\langle$`news`, $v\rangle$ if and only if v is an ancestor of cat on the category tree. We associate an authorization key $K(f)$ with every subscription filter f and an encryption key $K(e)$ with every event e. The authorization keys and the encryption keys satisfy the following properties:

- Given $K(f)$ it should be computationally easy to derive a key $K(e)$, if $v \in ancestor(cat)$.
- Given $K(f)$ it should be computationally infeasible to derive a key $K(e)$, if $v \notin ancestor(cat)$.

We construct keys that satisfy the above mentioned properties as follows. We map the authorization keys and encryptions into a common key space constructed using a category attribute key tree (CAKT for short). Given a subscription filter $\langle$`news`, $\ni$, $cat\rangle$, we use the CAKT to derive an authorization key $K(f)$ that corresponds to the category cat, denoted by K^{ont}_{cat}, where ont denotes the name of the ontology (in this example, ont = `news`). The CAKT enables one to easily (computationally) derive a key $K^{ont}_{cat'}$ from K^{ont}_{cat} if and only if $cat' \in ancestor(cat)$ in the category tree corresponding to the ontology ont. For any event e = $\langle$`news`, $v\rangle$, we encrypt the message with the encryption key $K(e) = K^{ont}_{v}$. By the construction of the category key tree it follows that $K(e)$ is easily derivable from $K(f)$ if and only if $v \in ancestor(cat)$.

Preliminaries. Given a category ontology, we map each category to a key tree identifier $ktid$. The root of the tree is assigned $ktid$ = $\emptyset$. We label the i^{th} child of a specialization with $ktid$ = ξ as $\xi \parallel i$. For the sake of simplicity we assume that CAKT is a binary category key tree such that each specialization has exactly two or zero children. However, the techniques discussed in this paper can be extended in a straightforward fashion to handle a case where different specializations in the category tree have different number of children. In our experimental section, we use an ontology tree wherein the number of children per tree node was randomly chosen between 2 to 4. Now we describe a technique to construct a CAKT, and then present algorithms for constructing subscriptions and publications using the CAKT.

Category Key Tree (CAKT). We derive a parent element ξ's key from its children elements $(\xi \parallel 0)$ and $(\xi \parallel 1)$ as follows: $K^{ont}_{\xi} = mix(blind(K^{ont}_{\xi\|0}), K^{ont}_{\xi\|1})) = mix(blind(K^{ont}_{\xi\|1}), K^{ont}_{\xi\|0}))$. There are several options for functions mix and $blind$. The function $blind$ is chosen such that given $blind(x)$ it is very hard to guess x. The function mix is chosen such that $mix(blind(x), y) = mix(blind(y), x)$.

The functions $blind$ and mix are defined based on Diffie-Hellman logical key hierarchy (DH-LKH) [21] as follows: $blind(x) = g^{H(x)} \bmod p$ and $mix(g^{H(x)} \bmod p, y) = g^{H(x)H(y)} \bmod p$. The parameter p is a large prime such that discrete log problem in the field Z_p is computationally intractable. The parameter g is a generator in field Z_p. Prime p and generator g are assumed to be system wide known parameters. Observe that $mix(g^{H(y)} \bmod p, x) = g^{H(y)H(x)} \bmod p = mix(g^{H(x)} \bmod p, y)$. Hence, K^{ont}_{ξ} is derived as $K^{ont}_{\xi} = g^{H(K^{ont}_{\xi\|0})H(K^{ont}_{\xi\|1})} \bmod p$ for some $\xi \in (0+1)^*$. We use the least significant 128 bits of the result as the actual key. For example, $K^{\texttt{news}}_{0} = g^{H(K^{\texttt{news}}_{00})H(K^{\texttt{news}}_{01})} \bmod p$.

Analogous to the category key tree, the pub-sub system also generates a *blinded key tree*. The *MS* is responsible for generating a blinded key tree $BK^{ont}_{\xi} = blind(K^{ont}_{\xi})$, for all key tree identifiers ξ in the CAKT. The blinded keys are required for a subscriber to generate the authorization keys for all the specializations that it is authorized for. However, if we use the OWH-LKH construction, the blinded key tree has to be kept a secret by the *MS*. Observe that given $blind(K^{ont}_{\xi\|0})$ and $blind(K^{ont}_{\xi\|1})$ any subscriber can generate $K^{ont}_{\xi} = blind(K^{ont}_{\xi\|0}) \oplus blind(K^{ont}_{\xi\|1}) = H(K^{ont}_{\xi\|0}) \oplus H(K^{ont}_{\xi\|1})$. Hence, if the blinded key tree were public, then any subscriber can derive the authorization keys for all non-leaf elements in the key tree. More concretely, by the hardness of the discrete log problem in the field Z_p it is computationally infeasible to derive a key K^{ont}_{ξ} using only its blinded key BK^{ont}_{ξ} or from its children blind keys $BK^{ont}_{\xi\|0}$ and $BK^{ont}_{\xi\|1}$.

For every leaf element with key tree identifier equal to $ktid$, the *MS* generates key $K^{ont}_{ktid} = KH_{K(w)}$ ($ont \parallel ktid$), where $K(w) = KH_{rk(MS)}(w)$ is the authorization key for the topic w, and $rk(MS)$ denotes the meta-service secret key. For example, the key for leaf element `secretNews1` (with $ktid$ = 00) under ontology ont = `news` and topic w = `cancerTrail` is derived as $K^{\texttt{news}}_{00} = KH_{K(\texttt{cancerTrail})}(\texttt{news} \parallel 00)$, where $K(\texttt{cancerTrail}) = KH_{rk(MS)}(\texttt{cancerTrail})$. For any non-leaf element on the key tree, the *MS* derives its key using the publicly

available blinded key tree for the topic w and ontology ont. Note that the ***MS*** does not have to store any extra confidential information to handle subscriptions; the key that corresponds to a given $ktid$ under a topic w can be efficiently computed on the fly. The ***MS*** spends only a one-time effort to generate a blinded key tree for every topic w and ontology ont.

Publication. The encryption keys for an event are constructed as follows:

$$\begin{aligned} e &= \langle\langle \texttt{topic}, w\rangle, \langle ont, cat\rangle, \langle \texttt{message}, msg\rangle\rangle \\ K(e) &= K^{ont}_{ktid(cat)} \oplus KP(P, w) \end{aligned}$$

For example, a publication e = $\langle\langle$`topic`, `cancerTrail`$\rangle$, $\langle$`news`, `classifiedNews1`$\rangle$, $\langle$`message`, $msg\rangle\rangle$ is encrypted as follows. P identifies the element `unclassifiedNews1` in the key tree has an identifier 0 (see Figure 2). P generates the encryption key $K(e)$ = $K_0^{\texttt{news}} \oplus KP(P, w)$.

Subscription. The authorization keys for a subscription filter are constructed as follows:

$$\begin{aligned} f &= \langle\langle \texttt{topic}, EQ, w\rangle, \langle ont, \ni, cat\rangle\rangle \\ K(f) &= K^{ont}_{ktid(cat)}, KS(w) \end{aligned}$$

Given a publication with msg with `ktid` = $ktid_\alpha$ a subscriber who has subscribed for `ktid` = $ktid_f$ does the following. The subscriber checks if $ktid_\alpha$ is a prefix of $ktid_f$. The subscriber uses this information to extract the suffix $b_0 b_1 \cdots b_{m-1}$ and derives the key $K^{ont}_{ktid_\alpha}$ from the key $K^{ont}_{ktid_\phi}$. Observe that any subscriber who possesses the key that corresponds to some element of the key tree can efficiently derive the keys for all its ancestors recursively as $K^{ont}_{\xi} = (BK^{ont}_{\xi \| b})^{H(K^{ont}_{\xi \| \overline{b}})} \ mod\ p$ for some $\xi \in (0+1)^*$, b = 0 or 1 and $\overline{b}$ denotes the bit complement of b. However, it is computationally infeasible for a subscriber to derive the keys corresponding to its children or siblings.

For example, given a publication with message msg encrypted with the key $K_0^{\texttt{news}}$ a subscriber S who possesses the key $K_{00}^{\texttt{news}}$ does the following. The subscriber extracts the publication's key tree identifier $ktid_\phi$ = 0 and its subscription's key tree identifier $ktid_\alpha$ = 00. Then, S identifies that element 0 is an ancestor of element 00. Then, S derives $K_0^{\texttt{news}} = (BK_{01}^{\texttt{news}})^{H(K_{00}^{\texttt{news}})} \ mod\ p = g^{H(K_{00}^{\texttt{news}})H(K_{01}^{\texttt{news}})} \ mod\ p$ (since, $BK_{01}^{\texttt{news}} = g^{H(K_0^{\texttt{news}})} \ mod\ p$). Recall that the blinded key tree $BK^{\texttt{news}}_{ktid}$ is made publicly available by the pub-sub system. Now, S can use $K_0^{\texttt{news}}$ to decrypt the message msg in the publication.

3.5 String Based Suffix and Prefix Matching

In this section, we present our key derivation algorithm for string based suffix/prefix matching. Given a string attribute, say `name`, we can construct a subscription filter f = $\langle$`name`, PF, $u\rangle$. The filter f matches any event e = $\langle$`name`, $v\rangle$ if and only if string u is a prefix of string v. We associate an authorization key $K(f)$ with every subscription filter f and an encryption key $K(e)$ with every event e. The authorization keys and the encryption keys satisfy the following properties:

- Given $K(f)$ it should be computationally easy to derive a key $K(e)$, if $u\ PF\ v$, that is, u is a prefix of v.
- Given $K(f)$ it should be computationally hard to derive a key $K(e)$, if u is not a prefix of v.

We construct keys that satisfy the above mentioned properties as follows. We map the authorization keys and encryption keys to the common key space using a string attribute key tree (SAKT for short). Given a subscription filter $\langle str, PF, u\rangle$, we use the SAKT to derive an authorization key that corresponds to the string u, denoted by K_u^{str}, where str denotes the name of the string attribute. The SAKT enables one to easily (computationally) derive a key K_v^{str} from K_u^{str} if and only if $u\ PF\ v$. For any event e = $\langle str, v\rangle$, we encrypt the message with $K(e)$ = K_v^{str}. By the construction of the string attribute key tree it follows that $K(e)$ is easily derivable from $K(f)$ if and only if $u\ PF\ v$. Our construction for string suffix matching is very similar to string prefix matching and will not be discussed separately.

String Attribute Key Tree (SAKT). We first present a technique to construct the SAKT. Given a string value u =

$u_0u_1 \cdots u_{k-1}$, where u_i denote characters in the string u, we construct a key K_u^{str} recursively as follows: $K_{u_0u_1 \cdots u_i}^{str} = H(K_{u_0u_1 \cdots u_{i-1}}^{str} \parallel u_i)$, where H denotes a one-way hash function. Let the symbol $\emptyset$ denote the null string. We derive the authorization key for the null string corresponding to the key tree as $K_{\emptyset}^{str} = KH_{K(w)}(str)$, where KH is a keyed pseudo-random function, $K(w) = KH_{rk(MS)}(w)$ is the authorization key for the topic w, and $rk(MS)$ denotes the *MS*'s secret key. An example topic would be w = `cancerTrail` and numeric attribute str = `name`. Note that $\parallel$ denotes string concatenation. For example, $K_{\texttt{a}}^{str}$ is derived as $K_a^{str} = H(K_{\emptyset}^{str} \parallel \texttt{a})$ and $K_{\texttt{an}}^{str}$ is derived as $K_{\texttt{an}}^{str} = H(K_{\texttt{a}}^{str} \parallel \texttt{b})$.

Publication. A publisher P constructs the encryption key for a numeric attribute event e as follows:

$$\begin{aligned} e &= \langle\langle \texttt{publisher}, P\rangle, \langle \texttt{topic}, w\rangle, \langle str, v\rangle, \langle \texttt{message}, msg\rangle\rangle \\ K(e) &= K_v^{str} \oplus KP(P, w) \end{aligned}$$

For example, given a publication e = $\langle\langle$`publisher`, $P\rangle$, $\langle$`topic`, `cancerTrail`$\rangle$, $\langle$`name`, `andy`$\rangle$, $\langle$`message`, $msg\rangle\rangle$ we construct $K(e) = K_{\texttt{andy}}^{\texttt{name}} \oplus KP(P, \texttt{cancerTrail})$.

Subscription. We construct an authorization key for a subscription as follows.

$$\begin{aligned} f &= \langle\langle \texttt{topic}, EQ, w\rangle, \langle str, PF, u\rangle\rangle \\ K(f) &= K_u^{str}, KS(w) \end{aligned}$$

Given a publication with string v a subscriber who has subscribed for a string u does the following. The subscriber checks if u is a prefix of v. If so, the subscriber derives the encryption key K_v^{str} from the authorization key K_u^{str}. Note that the generation of children keys from its parent's key is computationally efficient because it uses a fast one-way hash function. However, it is computationally infeasible for a subscriber to derive the keys corresponding to its ancestors or its siblings. For example, given a publication with v = `andy`, a subscriber who has subscribed for u = `a` decrypts the message msg in a publication as follows. Given v = `andy` and u = `a`, the subscriber first extracts the suffix `ndy`. Then, S derives $K_{\texttt{an}}^{str} = H(K_{\texttt{a}}^{str} \parallel \texttt{n})$, $K_{\texttt{and}}^{str} = H(K_{\texttt{an}}^{str} \parallel \texttt{d})$, and $K_{\texttt{andy}}^{str} = H(K_{\texttt{and}}^{str} \parallel \texttt{y})$.

3.6 Complex Subscriptions

We have so far presented EventGuard techniques to handle numeric attribute and category based subscriptions. However, we have so far dealt with subscriptions that consists of a topic and at most one constraint, say, f = $\langle\langle$`topic`, EQ, `cancerTrail`$\rangle$, $\langle$`age`, $\in$, (0, 15)$\rangle\rangle$. A complex subscription could consist of constraints combined using the $\wedge$ and $\vee$ Boolean operators. In general a complex filter is represented as a complex subscription f = $\langle\langle$`topic`, EQ, $w\rangle$, $B(sf_1, sf_2, \cdots, sf_l)\rangle$ where each sf_i is a simple filter (only one constraint) and B is a monotone Boolean expression. An example of a complex filter could be f=$\langle\langle$`topic`, EQ, `cancerTrail`$\rangle$, ($\langle$`age`, $\in$, (0, 15)$\rangle$ $\wedge$ $\langle$`gender`, EQ, `F`$\rangle$ $\wedge$ ($\langle$`news`, $\ni$, `secretNews1`$\rangle$ $\vee$ $\langle$`news`, $\ni$, `secretNews5`$\rangle$))$\rangle$. A matching event for the example subscription shown above could be e = $\langle\langle$`topic`, `cancerTrail`$\rangle$, $\langle$`age`, 9$\rangle$, $\langle$`gender`, `F`$\rangle$, $\langle$`news`, `classifiedNews1`$\rangle$, $\langle$`message`, $msg\rangle\rangle$. In this section we compose techniques presented in Sections 3.2, 3.3 and 3.4 to handle complex subscriptions.

Preliminaries. Given a complex subscription f = $\langle\langle$`topic`, EQ, $w\rangle$, $B(sf_1, sf_2, \cdots, sf_l)\rangle$, we express B in disjunctive normal form (DNF) [14] as $B = \bigvee_{j=1}^{nd} D_j$, where $D = \bigwedge_{j=1}^{nsf} sf_j$. We then divide f into nd complex filters $\{f_1, f_2, \cdots, f_{nd}\}$, where f_i = $\langle\langle$`topic`, EQ, $w\rangle$, $D_i(sf_1, sf_2, \cdots, sf_l)\rangle$. The subscriber now subscribes independently for each of these nd subscription filters. Note that this is equivalent to the original subscription on the filter f since, $B = \bigvee_{j=1}^{nd} D_j$. For example, we divide a complex filter f = $\langle\langle$`topic`, EQ, `cancerTrail`$\rangle$, ($\langle$`age`, $\in$, (0, 15)$\rangle$ $\wedge$ $\langle$`gender`, EQ, `F`$\rangle$ $\wedge$ ($\langle$`news`, $\ni$, `secretNews1`$\rangle$ $\vee$ $\langle$`news`, $\ni$, `secretNews5`$\rangle$))$\rangle$ into two filters f_1 = $\langle\langle$`topic`, EQ, `cancerTrail`$\rangle$, ($\langle$`age`, $\in$, (0, 15)$\rangle$ $\wedge$ $\langle$`gender`, EQ, `F`$\rangle$ $\wedge$ $\langle$`news`, $\ni$, `secretNews1`$\rangle$))$\rangle$ and f_2 = $\langle\langle$`topic`, EQ, `cancerTrail`$\rangle$, ($\langle$`age`, $\in$, (0, 15)$\rangle$ $\wedge$ $\langle$`gender`, EQ, `F`$\rangle$ $\wedge$ $\langle$`news`, $\ni$, `secretNews5`$\rangle$))$\rangle$. Note that the subscriber can subscribe for the filters f_1 and f_2 independently and receive all events e that match the filter f = $f_1 \vee f_2$. In the following portions of this section we describe techniques to derive keys for complex filters whose constraints include only the $\wedge$ operator. The concrete technique for constructing subscriptions and publications using

these derived keys is very similar to that discussed in numeric attribute based matching and category based matching and will be omitted in this section.

Publication. The encryption key for a complex event $e_i = \langle\langle \texttt{topic}, w\rangle, \langle name_1, value_1\rangle, \cdots, \langle name_l, value_l\rangle\rangle$ is constructed as follows: $K(e_i) = H(\bigoplus_{j=1}^{l} K(se_j))$, where $se_j = \langle\langle \texttt{topic}, w\rangle, \langle name_j, value_j\rangle\rangle$. For example, given an event $e = \langle\langle \texttt{topic}, \texttt{cancerTrail}\rangle, \langle \texttt{age}, 9\rangle, \langle \texttt{gender}, \texttt{F}\rangle, \langle \texttt{news}, \texttt{classifiedNews1}\rangle, \langle \texttt{message}, msg\rangle\rangle$, the key $K(e)$ is computed as follows. We break up e_i into three simple events $se_1 = \langle\langle \texttt{topic}, \texttt{cancerTrail}\rangle, \langle \texttt{age}, 9\rangle\rangle$, $se_2 = \langle\langle \texttt{topic}, \texttt{cancerTrail}\rangle, \langle \texttt{gender}, \texttt{F}\rangle\rangle$ and $se_3 = \langle\langle \texttt{topic}, \texttt{cancerTrail}\rangle, \langle \texttt{news}, \texttt{classifiedNews1}\rangle\rangle$. Then, we compute the encryption keys for the simple events using the techniques described in Sections 3.2, 3.3 and 3.4: $K(se_1) = K_{010}^{\texttt{age}}$ (since $ktid(9) = 010$), $K(se_2) = K_F^{\texttt{gender}}$ and $K(se_3) = K_0^{\texttt{news}}$ (since $ktid(\texttt{classifiedNews1}) = 0$). Finally, we derive $K(e) = H(K_{010}^{\texttt{age}} \oplus K_F^{\texttt{gender}} \oplus K_0^{\texttt{news}})$.

Subscription. Now we describe how an authorization key is constructed for a filter $f_i = \langle\langle \texttt{topic}, EQ, w\rangle, D_i(sf_1, sf_2, \cdots, sf_l)\rangle$. We divide the subscription filter f_i into l simple subscriptions of the form: $f_{ij} = \langle\langle \texttt{topic}, EQ, w\rangle, sf_j\rangle$. The set of authorization keys associated with f_i is $\{K(sf_1), K(sf_2), \cdots, K(sf_l)\}$, where $K(sf_j)$ is the authorization key for a subscription filter f_{ij} using the techniques described in Sections 3.2, 3.3 and 3.4. For example, given a subscription filter $f_1 = \langle\langle \texttt{topic}, EQ, \texttt{cancerTrail}\rangle, (\langle \texttt{age}, \in, (0, 15)\rangle \wedge \langle \texttt{gender}, EQ, \texttt{F}\rangle \wedge \langle \texttt{news}, \ni, \texttt{secretNews1}\rangle)\rangle$, we split into three simple filters: $f_{11} = \langle\langle \texttt{topic}, EQ, \texttt{cancerTrail}\rangle, \langle \texttt{age}, \in, (0, 15)\rangle\rangle$, $f_{12} = \langle\langle \texttt{topic}, EQ, \texttt{cancerTrail}\rangle, \langle \texttt{gender}, EQ, \texttt{F}\rangle\rangle$ and $f_{13} = \langle\langle \texttt{topic}, EQ, \texttt{cancerTrail}\rangle, \langle \texttt{news}, \ni, \texttt{secretNews1}\rangle\rangle$. We then associate the following authorization keys with filter f: $K(f_{11}) = K_0^{\texttt{age}}$ (since $ktid(0, 15) = 0$), $K(f_{12}) = K_F^{\texttt{gender}}$, and $K(f_{13}) = K_{00}^{\texttt{news}}$ (since $ktid(\texttt{secretNews1}) = 00$). Given an event $e = \langle\langle \texttt{topic}, \texttt{cancerTrail}\rangle, \langle \texttt{age}, 9\rangle, \langle \texttt{gender}, \texttt{F}\rangle, \langle \texttt{news}, \texttt{classifiedNews1}\rangle, \langle \texttt{message}, E_{K(e)}(msg)\rangle\rangle$, a subscriber derives key $K(e)$ as follows. The subscriber computes $K_{010}^{\texttt{age}}$ (since $ktid(9) = 010$) from $K_0^{\texttt{age}}$ (since $ktid(0, 15) = 0$) using the numeric attribute key tree for $\texttt{age}$, $K_0^{\texttt{news}}$ (since $ktid(\texttt{classifiedNews1}) = 0$) from $K_{00}^{\texttt{news}}$ (since $ktid(\texttt{secretNews1}) = 00$) using the category attribute key tree for $\texttt{news}$. The subscriber uses the authorization key $K_F^{\texttt{gender}}$ with the derived keys $K_{010}^{\texttt{age}}$ and $K_0^{\texttt{news}}$ to compute $K(e) = H(K_{010}^{\texttt{age}} \oplus K_F^{\texttt{gender}} \oplus K_0^{\texttt{news}})$. One can show that a subscriber can derive $K(e)$ from $K(f)$ if and only if the complex event e matches the complex filter f by our composing arguments presented in Sections 3.2, 3.3 and 3.4.

3.7 Performance Enhancements

In this section we present two key caching mechanisms to enhance the performance of our key derivation algorithms: temporal key cache and semantic key cache. In a temporal cache, a key is cached with a hope that it is reused in the near future. A semantic key cache extends the temporal key cache by exploiting the functioning of our key derivation algorithms to enhance the system's performance.

Temporal Key Cache. A temporal key cache exploits the temporal locality in the events received by a subscriber. Caching the encryption key $K(e)$ saves the cost of computing $K(e)$ from $K(f)$. We use a simple least recently used (LRU) based cache replacement policy to maintain the temporal key cache.

Semantic Key Cache. The key idea behind the semantic key cache is to extend a temporal key cache using specific properties of our key derivation algorithm. Given an event e, the semantic key cache selects a cached authorization key $K(f_{opt})$ such that it is most efficient to derive $K(e)$ from $K(f_{opt})$. The optimal authorization key $K(f_{opt})$ for deriving the encryption key $K(e)$ is determined as follows. Given a filter f and an event e we define a distance between them as $dist(f, e)$. If the event e does not match the filter f, then $dist(f, e) = \infty$. If the event e matches the filter f, then we define $dist(f, e)$ as the computational cost incurred in deriving $K(e)$ from $K(f)$. We compute the optimal filter f_{opt} as $f_{opt} = argmin_{f \in C}\, dist(f, e)$, where C denotes the temporal key cache and $f \in C$ denotes a subscription filter f whose key $K(f)$ is cached in the temporal key cache C.

For example, for some numeric attribute num, let us suppose that the cache C consists of three keys $K_{\emptyset}^{num}$, K_0^{num} and K_1^{num}. Now we choose $K(f_{opt})$ to derive $K(e) = K_{000}^{num}$ as follows. First we observe that K_{000}^{num} can be derived only from keys $K_{\emptyset}^{num}$ and K_0^{num}. Computing K_{000}^{num} from $K_{\emptyset}^{num}$ requires three hash computations, while that from K_0^{num} requires two hash computations. Hence, we choose $K(f_{opt}) = K_0^{w}$.

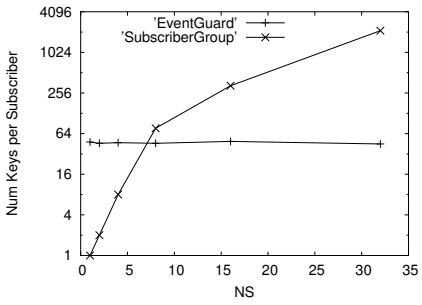

Figure 3: Num Keys per Subscriber

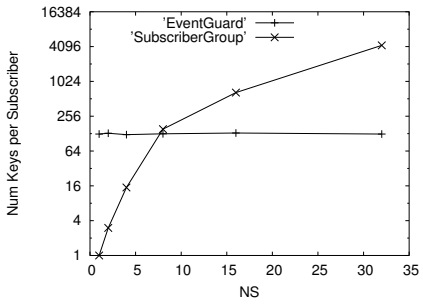

Figure 4: Num Keys per Publisher

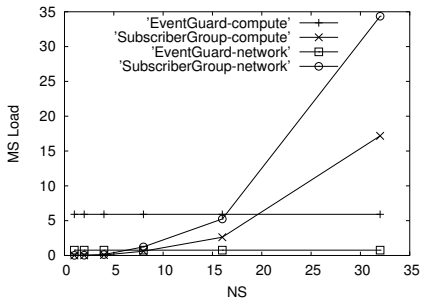

Figure 5: Meta-Service Load

4 Evaluation

4.1 Implementation Sketch

We have implemented our key management algorithms (EventGuard) on top of an unmodified Siena pub-sub core [6]. Siena is a content-based pub-sub system whose working is very similar to our reference model in Section 2. Hence, all EventGuard primitives can be directly mapped on to underlying Siena messages. A unique feature of our design is that the nodes in the pub-sub network can route messages as if they were *original* Siena messages. We have implemented *MS* as a stand-alone entity. The *MS* computes the authorization keys on the fly since the key derivation cost is very low. For a *MS* with limited computing power, one could use the semantic key caching mechanism to trade-off computing power with main memory utilization. For category trees, the *MS* pre-computes and stores the blinded key tree for every category in a publicly available MySQL Database [16]. Our implementation uses the following cryptographic algorithms. We use SHA1 for the hash function H, HMAC-SHA1 for the keyed hash function KH, and AES-128-CBC for the encryption algorithm E. For modular exponentiations in field Z_p, we use the standard exponentiation by squaring algorithm that computes the result in $O(\log_2 p)$ time.

4.2 Experimental Results

In this section, we present three sets of experiments on our prototype implementation. First we study the load on the meta-service. Second, we present measurements on the loss in throughput and the increase in latency in publications. Third, we show the effect of semantic key caching on the throughput and latency of the pub-sub system.

Experimental Setup. We used GT-ITM [30] topology generator to generate an Internet topology consisting of 63 nodes. The latencies for links were obtained from the underlying Internet topology generated by GT-ITM. The round trip times on these links varied from 24ms to 184ms with mean 74ms and standard deviation 50ms. The tree's root node acts as the publisher and its leaf nodes act as subscribers for this pub-sub network (32 subscribers and one publisher). We constructed complete binary tree topology using different number of nodes (0, 2, 6, 14, 30) and linked these nodes using open TCP connections to form the pub-sub network. The subscribers were uniformly distributed among all the leaf nodes. We ran our implementation on eight 8-processor servers (64 CPUs) (550 MHz Intel Pentium III Xeon processors running RedHat Linux 9.0) connected via a high speed LAN. We simulated the wide-area network delays obtained from the GT-ITM topology generator.

All experimental results presented in this section were averaged over 5 independent runs. Due to the lack of real workloads in this area, we had to use a synthetic workload. We simulated 128 topics, with the popularity of each topic varying according to a Zipf-like distribution [20]. Each subscriber subscribed for 32 topics chosen from the set of 128 topics using the Zipf distribution. Amongst 128 topics, 32 were numeric attributes, 32 were category trees, 32 were string attributes, and the rest 32 were simple topics. Numeric attributes had a range of size 256 units and a least count of 4 units. Hence, the number of elements in the numeric attribute tree was 127 and the height of the numeric attribute tree was 6. Category trees were for height 4 and number of children for each non-leaf element was chosen uniformly and randomly between 2 to 4. The average number of elements in a category tree was 82. The length of the string attributes were Zipf distributed between 1 and 8. Each publication message was assumed to be 256 Bytes long.

Number of Keys. Figure 3 shows the average number of keys maintained per subscriber as NS the number of subscribers varies. Recall that the `SubscriberGroup` approach uses group key management techniques on sub-

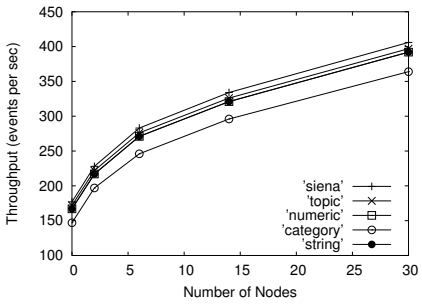

Figure 6: Throughput

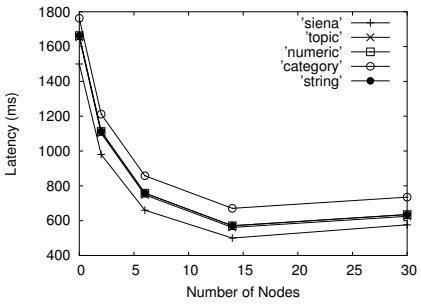

Figure 7: Latency

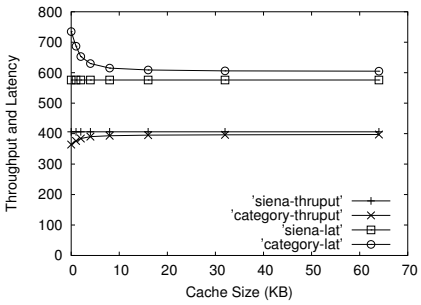

Figure 8: Key Caching

scriber groups [18]. EventGuard requires a small and constant number of keys per subscriber that is independent of NS. Even for 32 subscribers, the number of keys per subscriber using the `SubscriberGroup` approach is about 40 times larger than the EventGuard approach. EventGuard achieves significant reduction in the number of keys at the cost of running the key derivation algorithm on the subscriber. In later experiments we show that the cost of key derivation is very small compared to wide-area network latencies thereby making it easily affordable. Figure 4 shows the average number of keys maintained per publisher as NS the number of subscribers varies. The trends shown in Figure 4 are very similar to that in 3.

Meta-Service Load. Figure 5 shows the computing and network cost on the meta-service using `SubscriberGroup` based approach and `EventGuard`. Computing cost (measured in milliseconds) shows the average cost of group key management in `SubscriberGroup` and the cost of key derivation algorithm in `EventGuard` when a new subscriber joins the system. The cost incurred by the `SubscriberGroup` increases dramatically with NS (the number of subscribers that were already in the pub-sub network), while that incurred by the `EventGuard` approach is a small constant that is independent of NS. Networking cost (measured in Kbytes) shows the average cost of communicating the updated group key in `SubscriberGroup` and the cost of delivering the authorization keys in `EventGuard`. Similar to computing cost, `EventGuard` incurs a small and constant networking cost, while that of `SubscriberGroup` explodes with NS.

Throughput. We measured the throughput in terms of the maximum number of publications per second that can be handled by the pub-sub system. We measured the maximum throughput as follows. We engineered the publisher to generate publications at the rate of q publications per unit time. In each run of this experiment, the rate q was fixed. We monitored the number of outstanding publications required to be processed at every node. If at any node the number of outstanding publications monotonically increased for five consecutive observations, then we conclude that the node is saturated and the experiment aborted. We iteratively vary q across different experimental runs to identify the minimum value of $q_{min} = throughput$ such that some node in the pub-sub network is saturated. Figure 6 shows the maximum throughput versus the number of nodes in the pub-sub network. Observe that the throughput of the system increases with the number of nodes. This demonstrates the scalability of the pub-sub system and EventGuard's key management algorithms. Siena does not show much throughput difference between topic-based subscriptions, numeric attributes and categories. The throughput drop for numeric attributes is only marginally smaller than simple subscriptions since it only involves additional computations of hashes (< 1 μs); while that for category hierarchies is significantly larger because it involves a more expensive modular exponentiation (1.95 ms).

Latency. We measured latency in terms of the amount of time it takes from the time instant a publication is published till the time it is available to the subscriber (in plain-text). Figure 7 shows latency versus number of nodes. The latency is measured keeping the throughput of the system at its maximum. Observe that latency first decreases and then begins to increase with the number of nodes. Initially, latency decreases because the per-node processing cost decreases. However, as the number of nodes continues to increase, so does the diameter of the pub-sub network. Hence, distance between the publisher and subscriber (in terms of the number of pub-sub network hops) increases with the number of nodes. As in the case for throughput, Siena does not show any change in latency between topic-based subscriptions, numeric attributes and categories. Observe from Figure 7 that the increase in latency is very small. This is because the wide-area network latencies are of the order of 70ms; while the overhead for encryption/decryption and key derivation is relatively much smaller. Categories incur higher latency than numeric attributes because they require an expensive

modular exponentiation operation. Nonetheless the maximum increase in latency was lesser than 6%.

Semantic Key Cache. From our experiments on throughput and latency, we measured the overhead due to encryption/decryption and key derivation. We observed that the overhead due to encryption/decryption and key derivation for topics and numeric attributes was very low. However, the overhead of key generation is high for categories because it involves a modular exponentiation operation. In section 3.7, we have proposed a semantic key cache to improve the throughput and latency of the pub-sub system. Figure 8 shows the throughput and latency for category based matching in a pub-sub network with one publisher, 30 nodes and 32 subscribers for different values of cache size. Observe that when all authorization keys are cached, the encryption/decryption cost becomes the primary overhead for EventGuard. Using the semantic key caching mechanism the throughput of EventGuard was about 2.2% (as against 10.8% without caching) lower than Siena and the latency of EventGuard was about 1.5% (as against 5.7% without caching) higher than Siena (using a 64 KB cache). We also performed experiments wherein we cached keys for numeric attributes; however, the resulting throughput and latency improvement were very small (<0.5%).

5 Discussion

5.1 Issues

Matching Algorithms. In this paper we have demonstrated the functioning of EventGuard using three types of publication-subscription matching. One can use the techniques presented for numeric attribute matching to arbitrary prefix trees. One can use the techniques presented for category based matching for arbitrary directed acyclic graphs. However, the later class of matching algorithms incurs the overhead of having to maintain auxiliary data. Nonetheless, as we have shown in this paper, this auxiliary data may be made publicly available without compromising the security guarantees provided by the system. However, for classes of matching algorithms that have arbitrarily large key space and require auxiliary data, the size of auxiliary data may become a huge concern. As a part of our future work, we are characterizing key spaces and developing techniques to minimize the size of auxiliary data. One should note that in an extreme case, the matching algorithm could be any arbitrary function. It may be possible to handle arbitrary functions and provide complete confidentiality guarantees on routable attributes in an event using secure multi-party computation. However, the absence of efficient constructions for secure multi-party computations makes it infeasible to use them in practical systems.

Meta-Service Scalability. EventGuard uses a centralized meta-service for distributing the authorization keys. Our experiments show that the computing and network load incurred on our *MS* is significantly smaller than that incurred using a subscriber group approach. Our *MS* maintains only one 16 Byte key $rk(MS)$ and exports a very simple interface: subscribe and advertise. Unsubscribe is handled implicitly since a subscription has a valid life time (epoch); same holds for unadvertise. The *MS* is not involved in the publish operation; note that publish happens to the most common operation on the pub-sub network.

By our design, the *MS* can be easily replicated. The only secret that needs to be shared between the replicas is the 16 Byte master key $rk(MS)$. All the other pieces of shared data are public and read-only. This data includes least count for numeric attributes and category attribute trees and the auxiliary data required for key derivation. The fact that the replicas share only read-only data obviates the need for concurrency control amongst them. Additionally, we can vary the number of meta-service servers *on demand*, thereby permitting us to efficiently handle variable load (bursty load). Our experiments showed that the *MS* could handle up to 39 subscriptions per second. Using a semantic key cache with a modest 64 KBytes size, the *MS* can increase its throughput to 192 subscriptions per second. With three servers operating as the *MS* we obtained a throughput of 562 subscriptions per second using semantic key caching (3 * 192 = 576). Since the *MS* replicas require no synchronization and concurrency control, the meta-service shows a near linear scalability.

5.2 Related Work

Several pub-sub systems [6, 3, 7, 2] have been developed to provide highly scalable and flexible messaging support for distributed systems. Siena [6] and Gryphon [3] are large pub-sub system capable of content-aware routing. Scribe [7] is an anonymous P2P pub-sub system. Most work on pub-sub systems have focused on performance, scalability

and availability. Unfortunately, very little effort has been expended on studying the security aspects of these systems.

Wang et al. [26] analyze the security issues and requirements in a content-based pub-sub system. This paper identifies that the general security needs of a pub-sub application includes confidentiality, integrity and availability. More specifically they identify authentication of publications, integrity of publications, subscription integrity and service integrity as the key issues. The paper presents a detailed description of these problems in the context of a content-based pub-sub system, but fails to offer any concrete solutions.

Significant amount of work has been done in the field of secure group communication [1, 24, 25, 28, 15, 4, 5, 19] on multicast networks (survey [21]). Such systems leverage secure group-based multicast techniques and group key management techniques to provide forward and backward security, scalability and performance. A significant restriction with secure group communication is that the group membership is not as flexible as the subscription model used in pub-sub systems. In contrast, EventGuard permits flexible membership at the granularity of subscriptions. Also, EventGuard uses an overlay network and does not rely on IP multicast technology primarily because the IP multicast protocol has not yet been deployed at an Internet scale. EventGuard also differs from secure group communication protocols by associating authorization keys with subscription filters and encryption keys with events instead of associating keys with users or groups of users.

Opyrchal and Prakash [18] analyze secure distribution of events in a content-based pub-sub network from a group key management standpoint. They show that previous techniques for dynamic group key management fail in a pub-sub scenario since every event can potentially have a different set of interested subscribers. They use a key caching based technique that relies on subscription popularity to reduce the number of encryptions and to increase message throughput. However, their approach requires that the pub-sub network nodes (brokers) are completely trustworthy. We maintain complete confidentiality of secret attributes in an event from pub-sub nodes. For scalability we guarantee partial confidentiality on content routable attributes in the event.

Several techniques to safeguard a pub-sub network against message spoofing, spamming, and flooding attacks, addressing the issue of maintaining authentication and availability of publications and subscriptions are described in [23]. The paper also proposes techniques to construct a robust pub-sub network that is resilient to event dropping attacks by malicious pub-sub network nodes. Nonetheless, they focus entirely on guarding a pub-sub network from denial of service (DoS) attacks rather than attacks on event confidentiality and integrity.

Several authors have proposed reputation based feedback mechanism to guard a pub-sub network against dishonest publishers that publish malicious (invalid) events. A reputation based feedback mechanism [29] allows subscribers to assign scores to a publisher based on the *quality* of its publications. Eventually, subscribers would subscribe only to high quality publishers, and malicious publishers would run out of business.

6 Conclusion

We have presented EventGuard − secure, scalable and efficient key management algorithms for pub-sub systems. EventGuard protects the secret attributes in an event from the subscribers who have not subscribed for that event by encrypting publications such that only authorized subscribers of that event can read them. A unique feature of EventGuard is that it associates authorization keys with subscriptions and encryption keys with publications instead of associating keys with users or groups of users. EventGuard uses efficient and secure key derivation algorithms maintain the confidentiality guarantees while retaining performance & scalability and minimizing the overhead of key management. In this paper we have demonstrated the EventGuard approach using three common types of publication-subscription matching: topic or keyword based matching, numeric attribute based matching, and category based matching. We have also proposed semantic key cache based mechanisms to enhance the performance of our key derivation algorithms using a space-time trade off. We have presented a concrete implementation and a detailed evaluation of EventGuard on a pub-sub overlay network. Our experiments show that EventGuard meets the security goals while maintaining a low throughput (11%) and latency (6%) overhead on the pub-sub network.

Acknowledgement. This research is partially supported by NSF CNS, NSF ITR, an IBM SUR grant, and an HP Grant.

References

[1] K. Aguilera and R. Strom. Efficient atomic broadcast using deterministic merge. In *Proceedings of the 19th ACM PODC*, 2000.

[2] M. Aguilera, R. Strom, D. Sturman, M. Astley, and T. Chandra. Matching events in a content-based subscription system. In *Proceedings of the 18th ACM PODC*, 1999.

[3] G. Banavar, T. Chandra, B. Mukherjee, and J. Nagarajarao. An efficient multicast protocol for content-based publish subscribe systems. In *Proceedings of the 19th ICDCS*, 1999.

[4] R. Canetti, J. Garay, G. Itkis, and D. Micciancio. Multicast security: A taxonomy and some efficient constructions. In *Proceedings of the IEEE INFOCOM, Vol. 2, 708-716*, 1999.

[5] R. Canetti, T. Malkin, and K. Nissim. Efficient communication-storage tradeoffs for multicast encryption. In *Advances in Cryptology - EUROCRYPT. J. Stem, Ed. Lecture Notes in Computer Science, vol. 1599, Springer Verlag, pp: 459-474*, 1999.

[6] A. Carzaniga, D. S. Rosenblum, and A. L. Wolf. Design and evaluation of a wide-area event notification service. In *ACM Transactions on Computer System, 19(3):332-383*, 2001.

[7] A. K. Datta, M. Gradinariu, M. Raynal, and G. Simon. Anonymous publish/subscribe in p2p networks. In *Proceedings of IPDPS*, 2003.

[8] D. Eastlake and P. Jones. US secure hash algorithm I. http://www.ietf.org/rfc/rfc3174.txt, 2001.

[9] A. Fiat and M. Naor. Broadcast encryption. In *Springer Lecture Notes in Computer Science, pp: 480-491*, 1994.

[10] FIPS. Data encryption standard (DES). http://www.itl.nist.gov/fipspubs/fip46-2.htm.

[11] H. Harney and C. Muckenhirn. RFC 2093: Group key management protocol (GKMP) specification. http://www.rfc-archive.org/getrfc.php?rfc=2093.

[12] H. Harney and C. Muckenhirn. RFC 2094: Group key management protocol (GKMP) architecture. http://www.rfc-archive.org/getrfc.php?rfc=2094.

[13] H.Krawczyk, M. Bellare, and R. Canetti. HMAC: Keyed-hashing for message authentication. http://www.faqs.org/rfcs/rfc2104.html.

[14] Mathpages. Generating monotone boolean functions. http://www.mathpages.com/home/kmath094.htm.

[15] D. A. McGrew and A. T. Sherman. Key establishment in large dynamic groups using one-way function trees. In *Tech. Rep. No. 0755 (May), TIS Labs at Network Associates, Inc., Glenwood, MD*.

[16] MySQL. MySQL. http://www.mysql.com/.

[17] NIST. AES: Advanced encryption standard. http://csrc.nist.gov/CryptoToolkit/aes/.

[18] L. Opyrchal and A. Prakash. Secure distribution of events in content-based publish subscribe system. In *Proceedings of the 10th USENIX Security Symposium*, 2001.

[19] A. Perrig, D. Song, and J. D. Tygar. ELK: A new protocol for efficient large group key distribution. In *Proceedings of IEEE Symposium on Security and Privacy*, 2001.

[20] S. R. Qin Lv and S. Shenker. Can heterogeneity make gnutella scalable? In *Proceedings of the first International Workshop on Peer-to-Peer Systems*, 2002.

[21] S. Rafaeli and D. Hutchison. A survey of key management for secure group communication. In *Journal of the ACM Computing Surveys, Vol 35, Issue 3*, 2003.

[22] R. Rivest. The MD5 message-digest algorithm. http://www.ietf.org/rfc/rfc1321.txt, 1992.

[23] M. Srivatsa and L. Liu. Securing publish-subscribe overlay services using eventguard. In *Proceedings of ACM CCS*, 2005.

[24] M. Waldvogel, G. Caronni, D. Sun, N. Weiler, and B. Plattner. The versakey framework: Versatile group key management. In *IEEE Journal on Selected Areas in Communications (Special Issue on MiddleWare) 17, 9(Aug), 1614-1631*, 1999.

[25] D. Wallner, E. Harder, and R. Agee. Key management for multicast: Issues and architectures. In *RFC 2627*, 1999.

[26] C. Wang, A. Carzaniga, D. Evans, and A. L. Wolf. Security issues and requirements for internet-scale publish-subscribe systems. In *Proceedings of the 35th Hawaii International Conference on System Sciences (HICSS-35)*, 2002.

[27] Wikipedia. Ontology. http://en.wikipedia.org/wiki/Ontology.

[28] C. K. Wong, M. G. Gouda, and S. S. Lam. Secure group communications using key graphs. In *IEEE/ACM Transactions on Networking: 8, 1(Feb), 16-30*, 2000.

[29] L. Xiong and L. Liu. Peertrust: Supporting reputation-based trust for peer-to-peer electronic communities. In *Proceedings of IEEE TKDE, Vol. 16, No. 7*, 2004.

[30] E. W. Zegura, K. Calvert, and S. Bhattacharjee. How to model an internetwork. In *Proceedings of IEEE Infocom*, 1996.